

CS 47

Integers

Topics

- Basic operations
 - Addition, negation, multiplication
- Programming Implications
 - Consequences of overflow
 - Using shifts to perform power-of-2 multiply/divide

class04.ppt

CS 47 Spring, 2013

C Puzzles

- Taken from old exams
- Assume machine with 32 bit word size, two's complement integers
- For each of the following C expressions, either:
 - Argue that is true for all argument values
 - Give example where not true

Initialization

```
int x = foo();  
int y = bar();  
unsigned ux = x;  
unsigned uy = y;
```

- $x < 0 \Rightarrow ((x*2) < 0)$
- $ux \geq 0$
- $x \& 7 == 7 \Rightarrow (x \ll 30) < 0$
- $ux > -1$
- $x > y \Rightarrow -x < -y$
- $x * x \geq 0$
- $x > 0 \&\& y > 0 \Rightarrow x + y > 0$
- $x \geq 0 \Rightarrow -x \leq 0$
- $x \leq 0 \Rightarrow -x \geq 0$

- 2 -

CS 47 Spring 2013

C Puzzle Answers

- Assume machine with 32 bit word size, two's comp. integers
- *TMin* makes a good counterexample in many cases

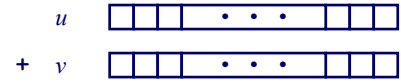
□ $x < 0$	$\Rightarrow$	$((x*2) < 0)$	False: <i>TMin</i>
□ $ux \geq 0$			True: $0 = UMin$
□ $x \& 7 == 7$	$\Rightarrow$	$(x \ll 30) < 0$	True: $x_1 = 1$
□ $ux > -1$			False: 0
□ $x > y$	$\Rightarrow$	$-x < -y$	False: $-1, TMin$
□ $x * x \geq 0$			False: 46341
□ $x > 0 \&\& y > 0$	$\Rightarrow$	$x + y > 0$	False: <i>TMax</i> , <i>TMax</i>
□ $x \geq 0$	$\Rightarrow$	$-x \leq 0$	True: $-TMax < 0$
□ $x \leq 0$	$\Rightarrow$	$-x \geq 0$	False: <i>TMin</i>

- 3 -

CS 47 Spring 2013

Unsigned Addition

Operands: w bits



True Sum: $w+1$ bits



Discard Carry: w bits

$UAdd_w(u, v)$



Standard Addition Function

- Ignores carry output

Implements Modular Arithmetic

$$s = UAdd_w(u, v) = u + v \bmod 2^w$$

$$UAdd_w(u, v) = \begin{cases} u + v & u + v < 2^w \\ u + v - 2^w & u + v \geq 2^w \end{cases}$$

- 4 -

CS 47 Spring 2013

Mathematical Properties

Modular Addition Forms an Abelian Group

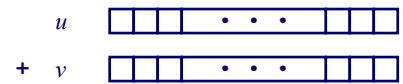
- Closed under addition
 $0 \leq \text{UAdd}_w(u, v) \leq 2^w - 1$
- Commutative
 $\text{UAdd}_w(u, v) = \text{UAdd}_w(v, u)$
- Associative
 $\text{UAdd}_w(t, \text{UAdd}_w(u, v)) = \text{UAdd}_w(\text{UAdd}_w(t, u), v)$
- 0 is additive identity
 $\text{UAdd}_w(u, 0) = u$
- Every element has additive inverse
 - Let $\text{UComp}_w(u) = 2^w - u, u > 0. \text{UComp}_w(0) = 0$
 $\text{UAdd}_w(u, \text{UComp}_w(u)) = 0$

- 5 -

CS 47 Spring 2013

Two's Complement Addition

Operands: w bits



True Sum: $w+1$ bits



Discard Carry: w bits

$\text{TAdd}_w(u, v)$



TAdd and UAdd have Identical Bit-Level Behavior

- Signed vs. unsigned addition in C:


```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v;
```
- Will give $s == t$

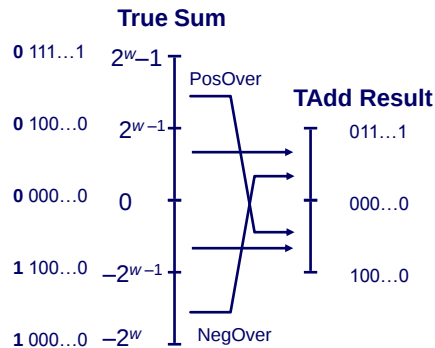
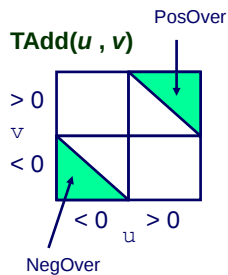
- 6 -

CS 47 Spring 2013

Characterizing TAdd

Functionality

- True sum requires $w+1$ bits
- Drop off MSB
- Treat remaining bits as 2's comp. integer



$$TAdd_w(u, v) = \begin{cases} u + v + 2^w & u + v < TMin_w \text{ (NegOver)} \\ u + v & TMin_w \leq u + v \leq TMax_w \\ u + v - 2^w & TMax_w < u + v \text{ (PosOver)} \end{cases}$$

- 7 -

CS 47 Spring 2013

Detecting 2's Comp. Overflow

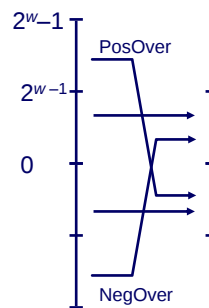
Task

- Given $s = TAdd_w(u, v)$
- Determine if $s = Add_w(u, v)$
- Example


```
int s, u, v;
s = u + v;
```

Claim

- Overflow iff either:
 - $u, v < 0, s \geq 0$ (NegOver)
 - $u, v \geq 0, s < 0$ (PosOver)
- $$ovf = (u < 0 == v < 0) \ \&\& \ (u < 0 != s < 0);$$



- 8 -

CS 47 Spring 2013

Mathematical Properties of TAdd

Isomorphic Algebra to UAdd

- $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
 - Since both have identical bit patterns

Two's Complement Under TAdd Forms a Group

- Closed, Commutative, Associative, 0 is additive identity
- Every element has additive inverse
Let $TComp_w(u) = U2T(UComp_w(T2U(u)))$
 $TAdd_w(u, TComp_w(u)) = 0$

$$TComp_w(u) = \begin{cases} -u & u \neq TMin_w \\ TMin_w & u = TMin_w \end{cases}$$

- 9 -

CS 47 Spring 2013

Multiplication

Computing Exact Product of w -bit numbers x, y

- Either signed or unsigned

Ranges

- Unsigned: $0 \leq x * y \leq (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
 - Up to $2w$ bits
- Two's complement min: $x * y \geq (-2^{w-1}) * (2^{w-1} - 1) = -2^{2w-2} + 2^{w-1}$
 - Up to $2w-1$ bits
- Two's complement max: $x * y \leq (-2^{w-1})^2 = 2^{2w-2}$
 - Up to $2w$ bits, but only for $(TMin_w)^2$

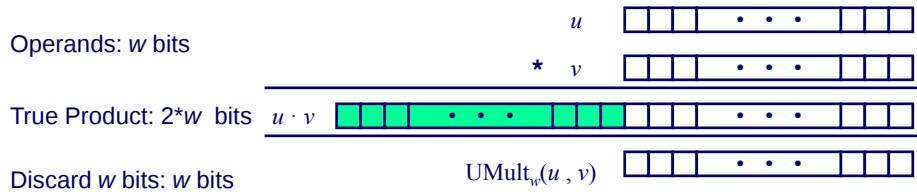
Maintaining Exact Results

- Would need to keep expanding word size with each product computed
- Done in software by “arbitrary precision” arithmetic packages

- 10 -

CS 47 Spring 2013

Unsigned Multiplication in C



Standard Multiplication Function

- Ignores high order w bits

Implements Modular Arithmetic

$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$

– 11 –

CS 47 Spring 2013

Unsigned vs. Signed Multiplication

Unsigned Multiplication

```
unsigned ux = (unsigned) x;  
unsigned uy = (unsigned) y;  
unsigned up = ux * uy
```

- Truncates product to w -bit number $up = \text{UMult}_w(ux, uy)$
- Modular arithmetic: $up = ux \cdot uy \bmod 2^w$

Two's Complement Multiplication

```
int x, y;  
int p = x * y;
```

- Compute exact product of two w -bit numbers x, y
- Truncate result to w -bit number $p = \text{TMult}_w(x, y)$

– 12 –

CS 47 Spring 2013

Unsigned vs. Signed Multiplication

Unsigned Multiplication

```
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
unsigned up = ux * uy
```

Two's Complement Multiplication

```
int x, y;
int p = x * y;
```

Relation

- Signed multiplication gives same bit-level result as unsigned
- `up == (unsigned) p`

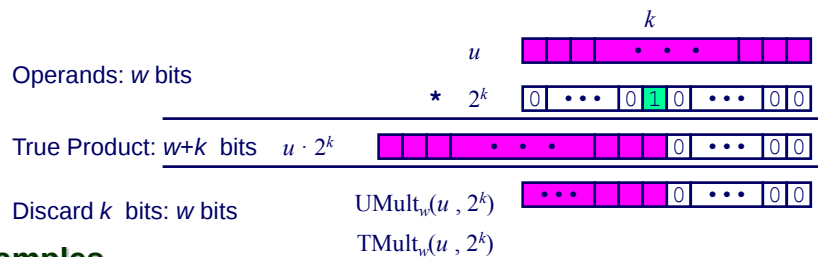
- 13 -

CS 47 Spring 2013

Power-of-2 Multiply with Shift

Operation

- `u << k` gives `u * 2k`
- Both signed and unsigned



Examples

- `u << 3 == u * 8`
- `u << 5 - u << 3 == u * 24`
- Most machines shift and add much faster than multiply
 - Compiler generates this code automatically

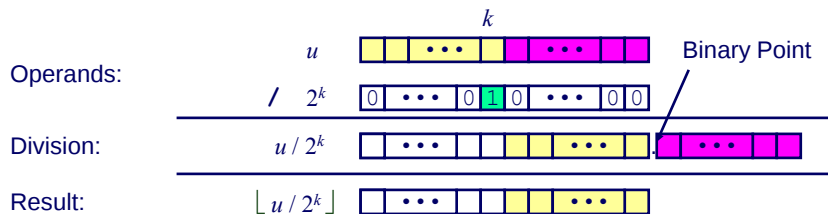
- 14 -

CS 47 Spring 2013

Unsigned Power-of-2 Divide with Shift

Quotient of Unsigned by Power of 2

- $u \gg k$ gives $\lfloor u / 2^k \rfloor$
- Uses logical shift



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

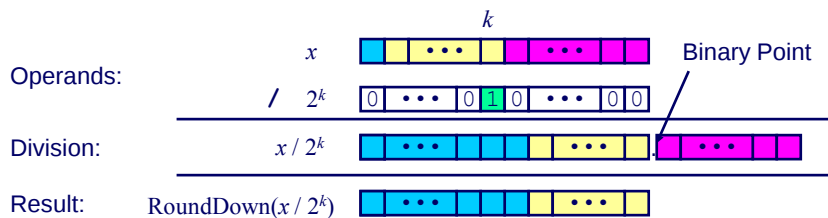
- 15 -

CS 47 Spring 2013

Signed Power-of-2 Divide with Shift

Quotient of Signed by Power of 2

- $x \gg k$ gives $\lfloor x / 2^k \rfloor$
- Uses arithmetic shift
- Rounds "wrong" direction when $x < 0$



	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	11100010 01001001
y >> 4	-950.8125	-951	FC 49	11111100 01001001
y >> 8	-59.4257813	-60	FF C4	11111111 11000100

- 16 -

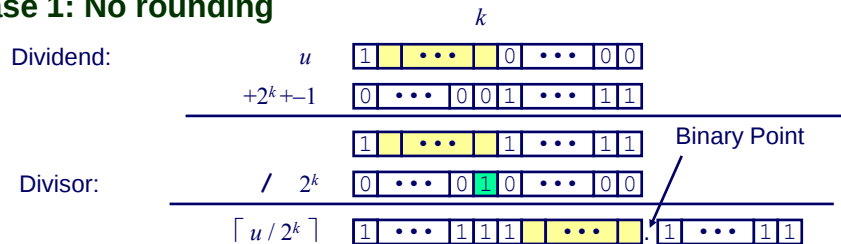
CS 47 Spring 2013

“Correct” Power-of-2 Divide

Quotient of Negative Number by Power of 2

- Want $\lceil x / 2^k \rceil$ (Round Toward 0)
- Compute as $\lfloor (x + 2^k - 1) / 2^k \rfloor$
 - In C: $(x + (1 << k) - 1) >> k$
 - Biases dividend toward 0

Case 1: No rounding



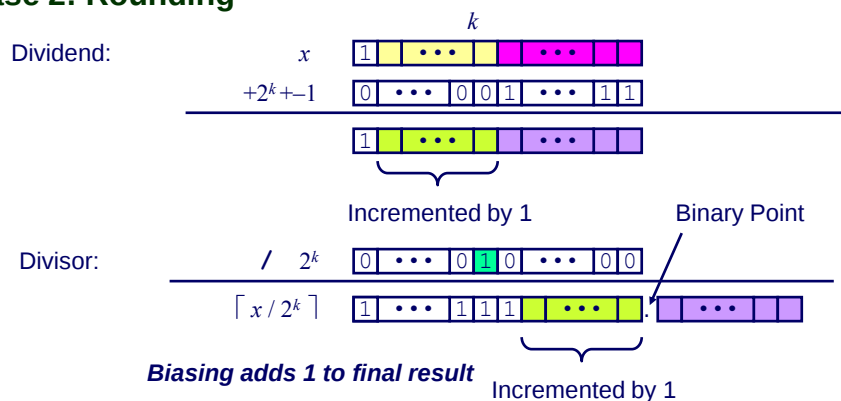
Biasing has no effect

- 17 -

CS 47 Spring 2013

“Correct” Power-of-2 Divide (Cont.)

Case 2: Rounding



- 18 -

CS 47 Spring 2013

Properties of Unsigned Arithmetic

Unsigned Multiplication with Addition Forms Commutative Ring

- Addition is commutative group
- Closed under multiplication
 $0 \leq \text{UMult}_w(u, v) \leq 2^w - 1$
- Multiplication Commutative
 $\text{UMult}_w(u, v) = \text{UMult}_w(v, u)$
- Multiplication is Associative
 $\text{UMult}_w(t, \text{UMult}_w(u, v)) = \text{UMult}_w(\text{UMult}_w(t, u), v)$
- 1 is multiplicative identity
 $\text{UMult}_w(u, 1) = u$
- Multiplication distributes over addition
 $\text{UMult}_w(t, \text{UAdd}_w(u, v)) = \text{UAdd}_w(\text{UMult}_w(t, u), \text{UMult}_w(t, v))$

- 19 -

CS 47 Spring 2013

Properties of Two's Comp. Arithmetic

Isomorphic Algebras

- Unsigned multiplication and addition
 - Truncating to w bits
- Two's complement multiplication and addition
 - Truncating to w bits

Both Form Rings

- Isomorphic to ring of integers mod 2^w

Comparison to Integer Arithmetic

- Both are rings
- Integers obey ordering properties, e.g.,
 $u > 0 \Rightarrow u + v > v$
 $u > 0, v > 0 \Rightarrow u \cdot v > 0$
- These properties are not obeyed by two's comp. arithmetic
 $TMax + 1 == TMin$

- 20 - $15213 * 30426 == -10030$ (16-bit words)

CS 47 Spring 2013